Introdução ao R

Vanderlei Júlio Debastiani (vanderleidebastiani@yahoo.com.br)

27 Maio 2020

Índice de conteúdo

1 Introdução

R é um ambiente de programação voltado para manipulação, análise de dados e apresentação gráfica. R é um projeto que está em constante desenvolvimento resultado de um esforço colaborativo com contribuições de todo o mundo. Foi inicialmente escrito por Robert Gentleman e Ross Ihaka do Departamento de Estatística da Universidade de Auckland, Nova Zelândia. Atualmente é desenvolvido pelo R Development Core Team.

O R possue uma estrutura de código aberto (*open source*) sendo gratuito com distribuição livre. Parte dos métodos estão implementados no ambiente básico do R (*R base*), mas muitos métodos estão implementadas em pacotes de funções adicionais (*packages*). Por ser uma linguagem fácil, como transparência em relação aos métodos e simples para qualquer um criar seu próprio pacote, o R atraiu um grande número de desenvolvedores ao longo do tempo. Atualmente existem mais de 13 mil pacotes disponíveis, sendo esse o grande diferencial do porquê usar o R.

Este texto trata aspectos básicos da linguagem R, como sintaxe, operações, tipos e estruturas de dados e importação e exportação de dados e resultados. Os código são brevemente comentados sendo que a ideia é que o leitor seja capaz de reproduzir os comandos aqui contidos, e busque mais informações sobre os argumentos extras nas páginas de ajuda de cada funçõe a fim de exercício. Ao final é apresentado um guia de ajuda rápida mostrando as principais palavras da sintaxe, operadores e funções da linguagem R apresentados aqui.

2 Sobre o R

2.1 Versões

As versões para Windows, Mac ou Linux podem ser baixadas no site oficial do projeto https://www.r-project.org. Nesse site podem serm encontradas mais informações sobre manuais, pacotes, documentação e novidades sobre o projeto. Além da versão básica existem outras versões, a principal delas é o *RStudio*. O RStudio é um ambiente de desenvolvimento integrado (IDE - *Integrated Development Environment*) para Windows, Mac e Linux. O programa é uma espécie de interface para o programa R e oferece um conjunto de ferramentas que facilitam a utilização. A versão de código aberto e gratuita pode ser baixada no site https://www.rstudio.com. A grande vantagem para quem estiver começando a utilizar o R é o recurso de autocompletar funções, argumentos e objetos durante a utilização.

2.2 Interface

O R funciona por meio de linhas de comandos. É preciso digitar algo para o programa executar. O programa recebe os comandos junto com os dados, executa as funções e retorna os resultados. Tanto o R quanto o RStudio apresentam algumas janelas, cada uma mostra um conjunto de informações. As principais delas são:

- **Console** - É a janela principal (a única que abre ao iniciar o R). Nela é possível digitar os comandos, vizualizar os resultados e mensagens de alerta e mensagens de erros. É possível navegar pelas funções já executadas, mas é difícil de editar os comandos nesta janela. Nesta janela a seta ($>$) indica que o R esta pronto para receber um comando; o sinal de mais ($+$) indica que o comando da linha anterior ainda não esta completo, faltando algo para o comando ser executado. A ausência de um desses dois simbólos ($>$ ou $+$) indica que o R ainda não finalizou o processo do comando anterior. Normalmente quando são mostrados os resultados de uma função um símbolos de cochetes ($[]$) com um valor numério (geralmente começa com *[1]*) é mostrado. Esse valor indica a posição (índice) do primeiro valor da respectiva linha.
- **Script** - Essa é a principal janela para edição dos comandos. Nela é possível editar facilmente os comandos, comentar as rotinas de análises e enviar para o console um ou mais comandos (Windows: Ctrl+R; Mac: Command+Enter; RStudio - Windows: Ctrl+Enter: RStudio - Mac: Command+Enter).
- **Help** - Janela onde são mostrados os textos de ajuda.
- **Plot** - Janela reservada para os resultados gráficos.
- **History** - Lista o histórico dos comandos já executados.
- **Package**s - Lista os pacotes (*packages*) instalados no computador.
- **Environment** - Painel exclusivo do RStudio que lista os objetos criados na área de trabalho (*workspace*).
- **Files** - Painel exclusivo do RStudio, que lista os arquivos do computador. É semelhante ao gerenciador de arquivos padrão do sistema operacional.

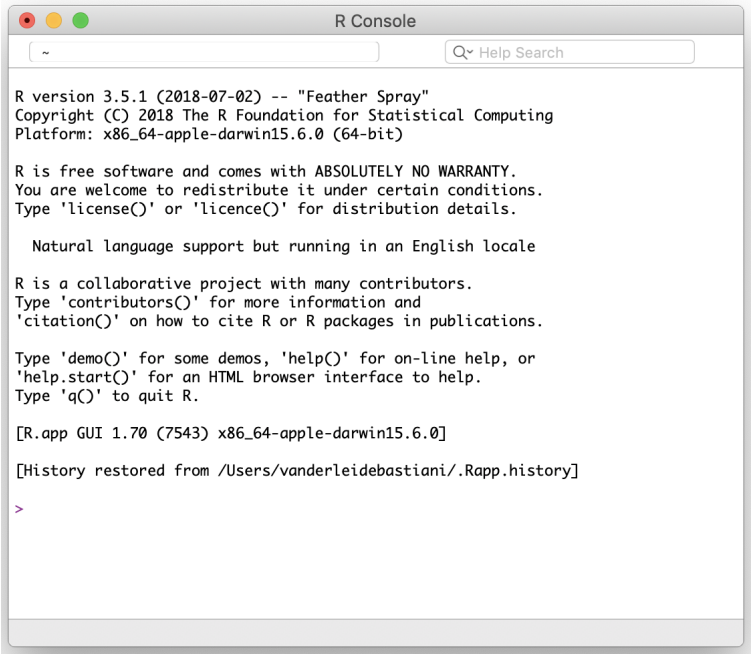


Figura 1: Janela do console

2.3 Arquivos

Os R gera alguns tipos de arquivos que podem ser salvos. As principais deles são:

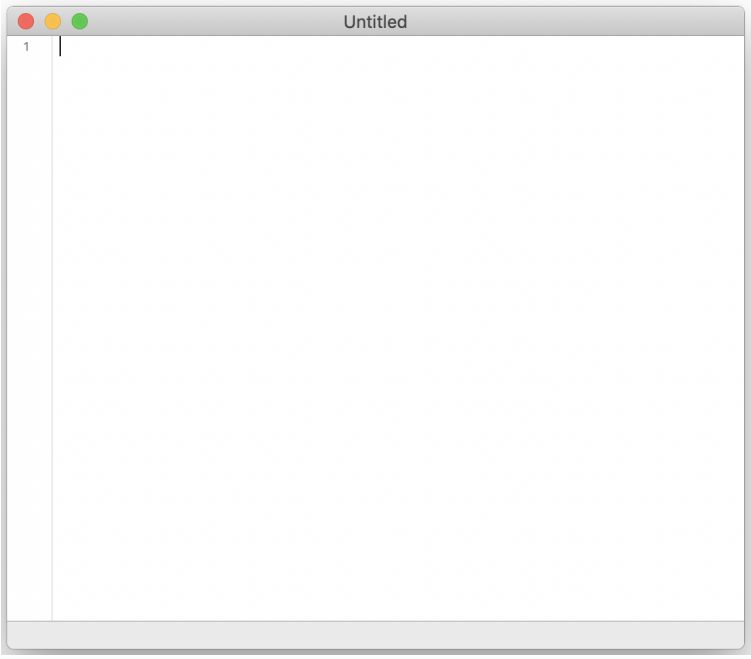


Figura 2: Janela do script

- **.R** - Usado para salvar códigos criados e rotinas de análises (*scripts*).
- **.RData** - Usado para salvar os objetos da área de trabalho (*workspace*).
- **.Rhistory** - Usado para salvar o histórico dos comandos executados (normalmente salvo automaticamente).
- **.Rproj** - Formato exclusivo do RStudio, serve para salvar todas as informações utilizadas anteriormente pelo RStudio de maneira simplificada.

2.4 Área de trabalho e diretório de trabalho

O R é uma linguagem de programação orientada a objeto, nele os objetos são atribuidos e salvos automaticamente de maneira temporária em uma área de trabalho (*workspace*). Ao abrir o R o *workspace* estará vazio, mas pode-se criar novos objetos para armazenar dados ou resultados de análises. Uma vez armazenados esses objetos podem ser exportados ou salvos em arquivos. Ao fechar o R, os objetos da área de trabalho serão apagados, exceto se foram anteriormente salvos. Ainda, o R sempre fica associado a uma pasta específica do seu computador, essa pasta será o diretório de trabalho (*working directory*). Os arquivos que estão neste diretório são fáceis de carregar e qualquer arquivo exportado será salvo no diretório caso não especificado o contrário. O diretório de trabalho pode ser alterado, sem qualquer mudanças nos objetos da área de trabalho.

```
getwd() # Mostrar o diretório de trabalho atual
dir() # Listar os arquivos do diretório
setwd() # Mudar o diretório de trabalho
```

É possivel salvar a área de trabalho inteira e carregar em outro momento.

```
ls() # Listar os objetos da área de trabalho
save.image("meu_workspace.RData") # Salvar área de trabalho
load("meu_workspace.RData") # Carregar área de trabalho
```

2.5 Pacotes

Como existem muitos pacotes para R só alguns deles são pré-instalados, os demais precisam ser instalados no computador. O procedimento de instalação precisa ser realizado apenas uma vez (além de atualização de tempos em tempos). Após instalados os pacotes estarão disponíveis para uso, mas é preciso carregar o pacote para utilização.

```
install.packages("bbmle") # Instalar o pacote chamado bbmle
library(bbmle) # Carrega o pacote, ou então...
require(bbmle) # ... também carrega o pacote
```

2.6 Detalhes importantes

Existem alguns detalhes que são importantes no R. Os principais são:

- **Diferenças entre caracteres maiúsculos e minúsculos** - Existem diferenças entre caracteres maiúsculos e caracteres minúsculos, então o caracter *A* é diferente do caracter *a*.
- **Separador decimal** - O separador decimal utilizado pelo R é o símbolo do ponto (.), o símbolo de vírgula (,) é reservado para separar diferentes objetos dentro de uma função. Para alterar essa opção é preciso configurar nas preferências gerais do sistema operacional.
- **Parênteses, colchetes e chaves** - Os símbolos de parênteses (*()*) servem para agrupar objetos dentro de uma função e os símbolos de chaves (*{}*) servem para agrupar funções dentro de outras funções. Os colchetes (*[]*) são utilizados para indexar objetos dentro de outros objetos. Como são símbolos utilizados para agrupar, eles devem ser abertos e fechados, ou seja, deve haver um símbolo para finalizar o agrupamento.
- **Caracteres especiais** - Caracteres especiais como por acentos e espaços devem ser evitados. Geralmente não são reconhecidos ou causam muitos problemas, pode-se usar os símbolos de ponto (.) ou underline (_) nos lugares dos espaços. Esses símbolos especiais podem se usados no formato de texto nos resultados finais, com por exemplo para exportar resultados gráficos.

3 Noções básicas

Ao digitar um comando o programa recebe os dados, executa as funções e retorna os resultados. Os dados de entrada podem ser fornecidos na hora de executar a função ou podem estar guardados em algum lugar da memória (geralmente na área de trabalho). Da mesma forma, os resultados das funções podem ser exibidos na tela ou salvados em algum objeto (geralmente na área de trabalho).

3.1 Calculadora

Primeiramente o R pode ser usado como uma calculadora.

```
2+2 # Soma
[1] 4
8-3 # Subtração
[1] 5
3*8 # Multiplicação
[1] 24
8/2 # Divisão
[1] 4
2^8 # Potências
[1] 256
(2+4)/7 # Prioridade de solução
[1] 0.8571429
```

Note que acompanhado do resultado é mostrado o *[1]*. Esse valor indica a posição (índice) do primeiro valor do resultado.

3.2 Estrutura das funções

Quase todas as funções do R seguem uma mesma estrutura. É preciso escrever o nome da função, abrir um parêntese, fornecer um ou mais argumentos para ela ser executada e fechar o parêntese.

$$função(argumento1 = valor1, argumento2 = valor2)$$

O número de argumentos de uma função é bastante varíavel, podendo variar de 0 a um grande número (geralmente de 1 a 3 argumentos). Por exmplo:

```
log(x = 8) # Logaritmo natural de 8
[1] 2.079442
rep(x = 1, times = 4) # Repetir o valor 1 quatro vezes
[1] 1 1 1 1
sum(1, 8, 79) # Soma de vários valores
[1] 88
```

Os valores/objetos para cada argumento precisam estar separados por vírgula (,). É preciso especificar o nome do argumento ou seguir a ordem especificada em cada função.

A função *log* precisa de dois argumentos:

$$log(x, base)$$

onde x é um valor númerico e *base* é a base que o algoritmo que será calculado.

```
log(x = 8, base = 2) # Logaritmo de 8 na base 2, especificando cada argumento
[1] 3
log(8, 2) # Logaritmo de 8 na base 2, seguindo a ordem dos argumentos na função
[1] 3
log(base = 2, x = 8) # Logaritmo de 8 na base 2, mudando a ordem dos argumentos. Note...
[1] 3
log(2, 8) # ... que o resultado é diferente se os argumentos não são especificados
[1] 0.3333333
```

Se verificarmos com detalhes os argumentos da função *log* vamos encontrar o seguinte:

$$log(x, base = exp(1))$$

o argumento *base* já está preestabelecido (*default*) sendo esse esse valor é *exp(1)*, ou seja, a função exponential do valor *1*, que é usado para para calcular o logaritmo natural.

```
log(x = 8, base = exp(1)) # Logaritmo de 8 na base natural. Isso é o mesmo...
[1] 2.079442
log(8) # ... se o argumento base não for especificado
[1] 2.079442
```

3.3 Ajuda

A função mais importante do R é a função *help* ou *?*. Com essa função é possível encontrar o arquivo de ajuda das funções. Esse arquivo geralmente contêm uma descrição do que a função exatamente faz, dos argumentos necessários, descrição dos resultados, referências e exemplos.

```
help(log) # Abre a ajuda da função log, ou então...
?log # ... também abre a ajuda
```

Os principais itens do arquivo de ajuda da função são:

- **Description** - Apresenta um resumo sobre o que a função faz.
- **Usage** - Mostra todos os argumentos, ordem e as opções preestabelecidas de cada um.
- **Arguments** - Explica cada argumento.
- **Details** - Detalhes sobre o uso da função, métodos e aplicação.
- **Value** - Explica cada um dos os resultados.
- **References** – Referências dos métodos.
- **See also** - Funções relacionadas.
- **Examples** - Mostra alguns exemplos que podem ser executados.

Os exemplos são bastante úteis pois mostram com as funções podem ser executadas. Geralmente eles funcionam com conjuntos de dados que já estão disponíveis dentro do R. Como isso, é possível verificar o funcionamento das funções antes mesmo de organizar seu conjunto de dados.

A função *args* mostra apenas os argumentos de uma função, o que pode ser útil em alguns casos.

```
args(log) # Mostra os argumentos da função log
function (x, base = exp(1))
NULL
```

Ainda é possivel pesquisar por assunto, usando a função *help.search* ou *??*. Qualquer documentação com o termo da pesquisa é retornada, incluindo funções de pacotes que não estejam carregados, mas que estejam instalados no computador.

```
help.search(mean) # Procura pelo termo mean, ou então...
??mean # ... faz a mesma busca
```

3.4 Comentários

É possível fazer comentários nos códigos e rotinas de análises usando o caracter sustenido (#).

```
# O sustenido é usado para escrever comantários...
2+2 #+2 ... todo o que vêem depois do símbolo não é reconhecido como comando
[1] 4
```

3.5 Atribuir objetos

Os objetos dentro do R podem ser salvos temporariamente na área de trabalho (*workspace*). Essa atribuição pode ser feita usando os símbolos <-, = ou -> e isso pode ser feito com qualquer tipo de objeto. Um resumo da estrutura de qualquer objeto no R por ser vista com a função *str*.

```
x <- 15 # O objeto x guarda o valor 15
x # Executando apenas o nome do objeto mostra o valor na tela, no caso 15
[1] 15
x <- 1000 # O objeto x guarda o novo valor. Note que o comando substituiu o valor anterior
x # Novo valor para x, no caso 1000. O valor anterior não pode ser mais recuperado
[1] 1000
X <- 10 # O objeto X guarda o valor 10
X # Valor 10. Note que há diferenças entre o objeto x (minúscula) e o objeto X (maiúsculo)
[1] 10
y <- sum(2, 9 , 43) # O objeto y guarda o resultado da soma
```

```
y # Executando apenas o nome do objeto mostra o resultado na tela
[1] 54
z <- y/19 # Operações podem ser feitas diretamente com os objetos
z # Resultado
[1] 2.842105
x <- z # Atribuir ao objeto x o valor do objeto z...
x # ... Desta forma o valor do objeto x é substituido pelo novo valor
[1] 2.842105
str(x) # Mostra um resumo de qualquer objeto do R
 num 2.84
```

3.6 Mostrar e remover objetos

Durante a execução do R vários objetos são salvos na área de trabalho (*workspace*). Para visualizar todos os objetos da área de trabalho é possível utilizar a função *ls* e para remover a função *rm*. Os objetos removidos não podem mais ser recuperados.

```
ls() # Listar dos objetos termporários da área de trabalho
[1] "x" "X" "y" "z"
rm(X) # Remover objeto X
rm(y) # Remover objeto y
rm(list = ls()) # Remover todos os objetos!
ls() # Área de trabalho vazia
character(0)
```

3.7 Símbolos especiais

Alguns caracteres e simbolos são reservados para uso especial dentro do R. Esses devem ser evitados ou não usados fora da proposta original. Os principais estão listados a seguir:

```
NA # Indeterminado (Not Avaiable)
NaN # Indeterminado (Not a Number)
Inf # Infinito
TRUE # Variável lógica para verdadeiro, ou abreviação T
FALSE # Variável lógica para falso, ou abreviação F
NULL # Usado para especificar algo nulo ou vazio
pi # Contante pi
```

Também existem algumas função que são composta apenas por uma letra. Nomes de objetos com essas letrasdevem ser evitados.

```
c # Concatenar valores
q # Fechar o R
t # Transpor uma matriz
C # Definer contrastes para um fator
D # Calcular derivadas
I # Mudar a classe de um fator
```

4 Tipos de dados

Os tipos de dados especificados dentro do R podem ser numéricos, caracteres ou lógicos, sendo que cada tipo é usado para um determinado fim.

4.1 Numérico (numeric)

A dados do tipo *numeric* são simplesmente número inteiros ou reais. Os números podem serem escritos normalmente ou com notação científica, utiliando o E (ou e).

```
x <- 4.5 # O . é o separador decimal
x
[1] 4.5
class(x)
[1] "numeric"
is.numeric(x) # Confere se o objeto é um da classe numeric
[1] TRUE
y<-1E4 # Notação científica
y
[1] 10000
class(y)
[1] "numeric"
is.numeric(y)
[1] TRUE
```

4.2 Caracter (character)

Os dados do tipo *character* são caracteres ou texto. São escritos obrigatoriamente entre aspas (*""* ou ”) e não são modificados. Os caracteres podem conter espaços ou acentos, mas isso deve ser utilizado para mostrar resultados finais, como nomes dos eixos em gráficos.

```
x <- "floresta" # Sempre entre aspas
x
[1] "floresta"
class(x)
[1] "character"
is.character(x) # Confere se o objeto é um da classe character
[1] TRUE
y <- "Diversidade (Simpson)" # Pode conter espaços ou acentos
y
[1] "Diversidade (Simpson)"
class(y)
[1] "character"
is.character(y)
[1] TRUE
```

4.3 Lógico (logical)

A dados do tipo *logical* são usados para as palavras que indicam verdadeiro ou falso, sendo as palavras *TRUE* ou *T* usadas para inidicar verdadeiro e as palavras palavras *FALSE* ou *F* para indicar falso. Não escritos obrigatoriamente com letras maiúsculas sem aspas. São utilizados para indicar opções onde há apenas duas opções ou como resultado de um teste lógico.

```
x <- TRUE # Não precisa estar entre aspas, é uma palavra especial no R
x
[1] TRUE
class(x)
[1] "logical"
is.logical(x) # Confere se o objeto é um da classe logical
```

```
[1] TRUE
y <- F # Pode ser abreviada
y
[1] FALSE
class(y)
[1] "logical"
is.logical(y)
[1] TRUE
```

5 Tipos de estruturas

Cada dados são organizados em diferentes estruturas. Cada uma delas apresenta um característica e pertence a uma determinada classe. A função *class* confere a classe de um objeto.

5.1 Vetores (vector)

Os vetores (*vector*) são objetos que armazenam um ou mais elementos, do tipo *numeric*, *character* ou *logical*. Os vetores podem ser criados com a função *c*, que concaneta objetos. Um vetor só suporta um único tipo de dados. Os vetores ainda possuem duas característica, uma delas é o comprimento que pode ser encontrado com a função *length* a segunda dela o nomes de cada elemento, acessada pela função *names*.

```
numeros <- c(2, 5, 6, 78, 2, 233) # A função c contena os números
numeros
[1]   2   5   6  78   2 233
class(numeros) # Confere a classe, só pode haver uma classe para todo o vetor
[1] "numeric"
is.vector(numeros) # Confere se o objeto é um vetor
[1] TRUE
length(numeros) # Mostra o comprimento do vetor
[1] 6
names(numeros) # Não há nomes
NULL
names(numeros) <- c("n1", "n2", "n3", "n4", "n5", "n6") # Atribuir nomes ao vetor numeros
names(numeros) # Os nomes do vetor
[1] "n1" "n2" "n3" "n4" "n5" "n6"
letras <- c("a", "b", "c", "d", "e", "f") # Classe character
letras
[1] "a" "b" "c" "d" "e" "f"
class(letras)
[1] "character"
logico <- c(F, T) # Classe logical
logico
[1] FALSE  TRUE
class(logico)
[1] "logical"
```

5.1.1 Indexação de vetores

Como um vetor só tem uma dimensão, para acessar um determinado valor do vetor é preciso fornecendo a posição do elemento no vetor entre colchetes (*[]*). Podem ser acessados um ou mais valores ao mesmo tempo. Se fornecer um valor com o sinal de menos na frente (-), todos os valores serão selecionados, exceto a posição em questão. Ainda, nos vetores com nomes é possível fazer a indexação usando o nome do elemento ou dados lógicos.

```
numeros[2] # Seleciona o segundo valor do vetor
n2
 5
numeros[c(3, 6, 1)] # Seleciona o terceiro, sexto e primeiro valor do vetor
 n3  n6  n1
  6 233   2
numeros[1:3] # Seleciona os primeiros três valores do vetor
n1 n2 n3
 2  5  6
numeros[12] # Índece fora dos limites, retorna um NA
<NA>
  NA
numeros[-3] # Seleciona tudo, exceto o terceiro valor
 n1  n2  n4  n5  n6
  2   5  78   2 233
numeros["n5"] # Seleciona o elemento com nome n5, no caso o quinto elemento
n5
 2
numeros[c(TRUE, FALSE, F, F, F, T)] # Seleciona o elemento o primeiro e último elemento
 n1  n6
  2 233
numeros[1] <- 12 # muda o valor do item 1 vetor numeros
```

5.2 Fatores (factor)

A classe *factor* serve para designar categorias para um vetor. Essa classe é semelhante a um vetor da classe *character*, mas tem importancia maior nas análises estatísticas já que designam um número finito (predefinido) de categorias. Tanto números quanto caracteres podem ser convertidos em fatores usando a função *factor*. Esses fatores podem não serem ordenados representado apenas diferentes categorias ou podem representar categorias ordenadas. A função *ordered* gera fatores ordenados, ou seja que as categorias tem diferentes níveis que são maiores uns que os outros.

```
fator1 <- factor(c("Co", "Co", "Co", "Tr", "Tr", "Tr")) # Fator a partir de caracteres
fator1
[1] Co Co Co Tr Tr Tr
Levels: Co Tr
class(fator1)
[1] "factor"
is.factor(fator1) # Confere se o objeto é um da classe factor
[1] TRUE
levels(fator1) # Confere as categorias do fator
[1] "Co" "Tr"
nlevels(fator1) # Confere o número de categorias do fator
[1] 2
is.ordered(fator1) # Confere se o fator é ordenado ou não
[1] FALSE
fator2 <- ordered(c(1, 1, 2, 2, 3, 3), levels = c(3, 2, 1)) # Cria um fator ordenado
fator2 # Note que a um símbolo de < para indicar a ordem
[1] 1 1 2 2 3 3
Levels: 3 < 2 < 1
levels(fator2) # Confere as categorias do fator
[1] "3" "2" "1"
is.ordered(fator2) # Confere se o fator é ordenado ou não
```

```
[1] TRUE
fator2 <- ordered(fator2, levels = c(2, 1, 3)) # É possível alterar a ordem dos níveis
fator2
[1] 1 1 2 2 3 3
Levels: 2 < 1 < 3
```

5.2.1 Indexação de fatores

A indexação dos fatores funciona exatamente da mesma maneira que os vetores.

```
fator1[2] # Seleciona o segundo valor do fator
[1] Co
Levels: Co Tr
fator1[2] <- "Tr" # mudar de fator
fator1 # níveis permanecem
[1] Co Tr Co Tr Tr Tr
Levels: Co Tr
fator1[1] <- "z" # fator inválido
Warning in `[<-.factor`(`*tmp*`, 1, value = "z"): invalid factor level, NA generated
fator1 # NA é gerado
[1] <NA> Tr   Co   Tr   Tr   Tr
Levels: Co Tr
```

5.3 Tabelas de dados (data.frame)

Tabelas de dados podem ser organizadas em data.frames usando a função *data.frame*. Os data.frames são tabelas que armazenam um ou mais vetores de dados. Por essas características eles possuem duas dimensões, linhas e colunas. Cada coluna, será um vetor, podendo portando armazenar vetores de diferentes tipos (*numeric*, *character* ou *logical*), entretanto o comprimento de todos vetores de um data.frame deve ser igual. Os data.frames possuem duas dimensões que podem ser encontradas com a função *dim*, *nrow* e *ncol*. Os nomes das linhas e colunas podem ser acessada pelas função *rownames* e *colnames* respectivamente. As funções *str*, *head* e *tail* mostram parte do data.frame para que sejá possível conferir os dados sem a nescessidade de visualizar a tabela completa.

```
dados <- data.frame(valores = numeros, letras, categorias = y) # Combinar vetores em ...
dados # ... um data.frame. Sempre por colunas
   valores letras categorias
n1      12      a      FALSE
n2       5      b      FALSE
n3       6      c      FALSE
n4      78      d      FALSE
n5       2      e      FALSE
n6     233      f      FALSE
str(dados) # Resumo do data.frame
'data.frame':   6 obs. of  3 variables:
 $ valores   : num  12 5 6 78 2 233
 $ letras    : chr  "a" "b" "c" "d" ...
 $ categorias: logi  FALSE FALSE FALSE FALSE FALSE FALSE
head(dados, n = 3) # Mostra as 3 primeiras linhas
   valores letras categorias
n1      12      a      FALSE
n2       5      b      FALSE
n3       6      c      FALSE
tail(dados, n = 2) # Mostra as 2 últimas linhas
```

```
   valores letras categorias
n5       2      e      FALSE
n6     233      f      FALSE
dim(dados) # Dimensões do data.frame, primeiro linhas o depois colunas
[1] 6 3
nrow(dados) # Número de linhas
[1] 6
ncol(dados) # Número de colunas
[1] 3
rownames(dados) # Nomes das linhas
[1] "n1" "n2" "n3" "n4" "n5" "n6"
colnames(dados) # Nomes das colunas
[1] "valores"    "letras"     "categorias"
```

5.3.1 Indexação de data.frames

Os data.frames possuem duas dimensões, para acessar um determinado valor é preciso fornecer a posição na linha e da coluna. Os valores são fornecidos entre colchetes (*[,]*), sepadaos por vírgula, sendo o primeiro valor a posição da linha e o segundo a posição da coluna. Como nos vetores podem ser acessados um ou mais linhas e colunas ao mesmo tempo e excluir da seleção determinadas linhas e colunas utilizando o sinal de menos (-) antes da posição. Além disso, é possível fazer a indexação das linhas e colunas usando nome entre aspas (*""*), ainda as colunas podem ser acessadas com símbolo cifrão (*$*). Se for selecionado apenas uma coluna, o R automaticamente transforma a seleção da classe *data.frame* para a classe *vector*. Para evitar que isso acontece é preciso fornecer um terceiro argumento dentro dos colchetes *drop = FALSE* (*[, , drop = FALSE]*).

```
dados[2, 1] # Valor da linha 2 e coluna 1
[1] 5
dados[c(1,4), 2:3] # Linhas 1 e 4 das colunas 2 e 3
   letras categorias
n1      a      FALSE
n4      d      FALSE
dados[2,] # Linha 2 com todas as colunas
   valores letras categorias
n2       5      b      FALSE
dados[, 2] # Coluna 2 com todas as linhas
[1] "a" "b" "c" "d" "e" "f"
dados[, 2, drop = FALSE] # Coluna 2 com todas as linhas, mantendo a estrutura de tabela
   letras
n1      a
n2      b
n3      c
n4      d
n5      e
n6      f
dados[, -3] # Seleciona tudo, exceto a terceira coluna
   valores letras
n1      12      a
n2       5      b
n3       6      c
n4      78      d
n5       2      e
n6     233      f
dados[, "letras"] # Seleciona a coluna chamada letras
```

```
[1] "a" "b" "c" "d" "e" "f"
dados["n1" ,] # Seleciona a linha chamada n1
   valores letras categorias
n1      12      a      FALSE
dados$valores # Seleciona a coluna chamada valores
[1]  12   5   6  78   2 233
dados$novo.vetor <- c("x", "y", "z", "x", "y", "z") # Adicionar vetor...
str(dados) # ... nesse caso a classe permanece
'data.frame':   6 obs. of  4 variables:
 $ valores   : num  12 5 6 78 2 233
 $ letras    : chr  "a" "b" "c" "d" ...
 $ categorias: logi  FALSE FALSE FALSE FALSE FALSE FALSE
 $ novo.vetor: chr  "x" "y" "z" "x" ...
dados$novo.vetor <- NULL # Deletar um vetor do data.frame (apenas colunas)
dados
   valores letras categorias
n1      12      a      FALSE
n2       5      b      FALSE
n3       6      c      FALSE
n4      78      d      FALSE
n5       2      e      FALSE
n6     233      f      FALSE
```

5.4 Matrizes (matrix)

Matrizes, da classe *matrix*, são praticamente iguais aos data.frames usando a função *matrix* ou com a combinação de vetores com as função *cbind* e *rbind* (concatena os vetores por coluna ou por linhas respectivamente). A diferença fundamental são que as matrizes só podem armazenar um tipo de dado, geralmente dados numéricos. Matematicamente isso é bastante importante pois se espera que as linhas e colunas das matrizes tenham relação umas com as outras. Como os data.frames, as matrizes possuem duas dimensões que podem ser encontradas com a função *dim*, *nrow* e *ncol*. Os nomes das linhas e colunas podem ser acessada pelas função *rownames* e *colnames* respectivamente. As funções *str*, *head* e *tail* também funcionam com matrizes. Além disso, as matrizes podem ser transpostas com a função *t*.

```
matriz <- matrix(c(1, 2, 3, 11, 12, 13), nrow = 3, ncol = 2) # Matriz, 3 x 2
matriz
     [,1] [,2]
[1,]    1   11
[2,]    2   12
[3,]    3   13
str(matriz) # Resumo da matriz
 num [1:3, 1:2] 1 2 3 11 12 13
head(matriz, n = 1) # Mostra a primeira linha
     [,1] [,2]
[1,]    1   11
tail(matriz, n = 1) # Mostra a última linha
     [,1] [,2]
[3,]    3   13
dim(matriz) # Dimensões da matriz, primeiro linhas o depois colunas
[1] 3 2
nrow(matriz) # Número de linhas
[1] 3
ncol(matriz) # Número de colunas
```

```
[1] 2
rownames(matriz) # Nomes das linhas, nesse caso NULL
NULL
colnames(matriz) # Nomes das colunas, nesse caso NULL
NULL
t(matriz) # Transpor a matriz
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]   11   12   13
matriz.por.colunas <- cbind(numeros, letras) # Concatenados por colunas
matriz.por.linhas <- rbind(numeros, letras) # Concatenados por linhas
matriz.por.colunas # Note que os números ficaram entre aspas, indicando...
   numeros letras
n1 "12"    "a"
n2 "5"     "b"
n3 "6"     "c"
n4 "78"    "d"
n5 "2"     "e"
n6 "233"   "f"
matriz.por.linhas # ... que foram convertidos em caracteres
        n1   n2  n3  n4   n5  n6
numeros "12" "5" "6" "78" "2" "233"
letras  "a"  "b" "c" "d"  "e" "f"
```

5.4.1 Indexação de matrizes

A indezação de matrizes funciona exatamente igual a indexação dos data.frames.

```
matriz[1, 1] # Valor da linha 1 e coluna 1
[1] 1
matriz[1, 1, drop = FALSE] # Valor da linha 1 e coluna 1, mantendo a estrutura da matriz
     [,1]
[1,]    1
matriz[-3, , drop = FALSE] # Seleciona tudo, exceto a terceira linha
     [,1] [,2]
[1,]    1   11
[2,]    2   12
matriz[3, 1] <- 0 # Nada pode ser deletado, apenas alterado o valor (incluindo NA)
matriz <- matriz[,c(2, 1)] # Mudar a ordem das colunas
matriz
     [,1] [,2]
[1,]   11    1
[2,]   12    2
[3,]   13    0
```

5.5 Listas (list)

Listas são objetos que podem armazenam quaisquer outros tipos de objetos, são pertencentes as classe *list*. As listas são criadas com a função *list* e por serem objetos bastante genérios e versáteis muitas vezes são usadas para armazenar os resultados de funções. As listas possuem apenas uma dimensão, que pode ser acessada pela função *length*, além disso, podem ter nomes, acessados pela função *names* e um resumo pode ser visto pela função *str*.

```
minha.lista <- list(matriz, 1, fator1) # Lista com objetos variados
minha.lista
[[1]]
     [,1] [,2]
[1,]   11    1
[2,]   12    2
[3,]   13    0

[[2]]
[1] 1

[[3]]
[1] <NA> Tr   Co   Tr   Tr   Tr
Levels: Co Tr
str(minha.lista) # Estrutura da lista
List of 3
 $ : num [1:3, 1:2] 11 12 13 1 2 0
 $ : num 1
 $ : Factor w/ 2 levels "Co","Tr": NA 2 1 2 2 2
length(minha.lista) # Mostra o comprimento do vetor
[1] 3
names(minha.lista) <- c("A", "B", "C") # Atribui nomes a lista
minha.lista
$A
     [,1] [,2]
[1,]   11    1
[2,]   12    2
[3,]   13    0

$B
[1] 1

$C
[1] <NA> Tr   Co   Tr   Tr   Tr
Levels: Co Tr
```

5.5.1 Indexação de listas

A indezação de listas funciona praticamente igual a indexação dos vetores, com a diferença que existem dois modelos de idexação. Se usado apenas um par de colchetes (*[]*) uma sublista é retornada, para indexar o conteúdo da lista é preciso de dois colchetes (*[[ ]]*). Além disso, as listas com nomes podem ser indexadas usando os nomes entre aspas (*""*) ou com o símbolo do cifrão (*$*). Os objetos dentro de uma lista podem ser acessados de maneira convencionar, sendo possível armazenar uma lista dentro de outra.

```
minha.lista[1] # Acessar primeira sublista
$A
     [,1] [,2]
[1,]   11    1
[2,]   12    2
[3,]   13    0
minha.lista[[1]] # Primeiro objeto da lista
     [,1] [,2]
[1,]   11    1
[2,]   12    2
```

```
[3,]   13    0
minha.lista[["C"]] # Objeto com o nome C na lista
[1] <NA> Tr   Co   Tr   Tr   Tr
Levels: Co Tr
minha.lista$A # Objeto com o nome A na lista
     [,1] [,2]
[1,]   11    1
[2,]   12    2
[3,]   13    0
minha.lista[[1]][3, 2] # Primeiro objeto da lista (uma matriz), linha 3, coluna 2
[1] 0
minha.lista[c(2:3)] # Acessar várias sublistas. minha.lista[[c(2:3)]] não funciona
$B
[1] 1

$C
[1] <NA> Tr   Co   Tr   Tr   Tr
Levels: Co Tr
minha.lista[["C"]] <- NULL # Deletar item da lista. Mesmo que minha.lista["C"]
minha.lista$novoitem <- 1:3 # Adicionar item a lista
minha.lista
$A
     [,1] [,2]
[1,]   11    1
[2,]   12    2
[3,]   13    0

$B
[1] 1

$novoitem
[1] 1 2 3
```

5.6 Conversão entre tipos e estruturas de dados

As dados e estruturas podem ser convertidas em outras tipos ou classes utilizando as funções *as.numeric*, *as.character*, *as.logical* e *as.factor*, *as.ordered*, entre outras. É preciso ter cuidado para não fazer uma conversão que não faça sentido.

```
x <- as.character(numeros) # Classe numeric para character
x
[1] "12"  "5"   "6"   "78"  "2"   "233"
x <- as.character(fator1) # Classe factor para character
x
[1] NA   "Tr" "Co" "Tr" "Tr" "Tr"
x <- as.numeric(letras) # Classe character para numeric não faz sentido
Warning: NAs introduced by coercion
x
[1] NA NA NA NA NA NA
x <- as.numeric(fator1) # Classe factor para numeric
x # Nesse caso é preciso ter cuidado pois o número da categoria fica como numeric
[1] NA  2  1  2  2  2
x <- c(0, 1, 0) # Vetor apenas de 0 e 1...
```

```
x
[1] 0 1 0
x <- as.logical(x) # ... pode ser convertido em logical
x # Os valores de 0 ficam como FALSE e valores defetentes de 0 ficam TRUE
[1] FALSE  TRUE FALSE
```

6 Entrada de dados

Existem várias maneiras de entrar com dados no R. As mais simples e práticas são utilizando planilhas de dados no formato .csv (*Comma-separated values*) ou .txt (separado por tabulações). Esses dois formatos são arquivos de texto simples usado para armazenar tabelas de dados, eles podem ser lidos em qualquer programa de edição de texto e podem ser salvos diretamente os programas de edição de planilhas. O .csv apresentar como separador entre tabelas a vírgula (,) já o .txt separado por tabulações o símbolo de tab. As duas principais funções para carregar os dados são *read.csv* e *read.table*, sendo que por padrão o símbolo de separação de colunas é vírgula para .csv e tab para .txt. Os principais argumentos destas funções são relacionados aos nomes das das linhas e colunas. Se a primeira linha do arquivo de dados for composta pelos nomes das colunas o argumentos *header* das funções deve ser verdadeiro. Já se a primeira coluna do arquivo contêm os nomes das linhas o argumento *row.names* deve ser igual a 1, indicando da coluna que contêm os nomes.

Para o evitar problemas é importante evitar uso de caracteres especiais, como espaços, acentos, células em branco. Use os símbolos de ponto (.) ou underline (_) nos lugares dos espaços e *NA* para indicar células faltantes no arquivo de dados. O arquivo .txt é mais difícil e exigente quanto a formatação que .cvs que é mais versátil e fácil para entrar com dados.

6.0.1 Descrição dos dados

O arquivo *eidat.csv* são 10 observações e 4 variáveis de dados simulados para ilustrar modelos de inferência ecológica. O arquivo *PlantGrowth.csv* são 30 observações e 2 variáveis de dados de resultados de um experimento para comparar os rendimentos obtidos sob controle e dois tratamentos. Os dados são disponibilizados em https://vincentarelbundock.github.io/Rdatasets/datasets.html

```
eidat <- read.csv("eidat.csv", row.names = 1, header = TRUE)
eidat # Conferir se os dados foram carregados corretamente
    x0   x1   t0   t1
1  200 3911 2850 1261
2  162 2636 1541 1257
3  206 2460 1091 1575
4  213 1654  517 1350
5  209  637  163  683
6  190 1911  216 1885
7  206 3460  226 3440
8  190  715  102  803
9  183 2058  126 2115
10 189 2658  138 2709
str(eidat)
'data.frame':   10 obs. of  4 variables:
 $ x0: int  200 162 206 213 209 190 206 190 183 189
 $ x1: int  3911 2636 2460 1654 637 1911 3460 715 2058 2658
 $ t0: int  2850 1541 1091 517 163 216 226 102 126 138
 $ t1: int  1261 1257 1575 1350 683 1885 3440 803 2115 2709
colnames(eidat)
[1] "x0" "x1" "t0" "t1"
rownames(eidat)
```

```
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
plantas <- read.csv("PlantGrowth.csv", row.names = 1, header = TRUE,
                    stringsAsFactors = TRUE)
str(plantas)
'data.frame':   30 obs. of  2 variables:
 $ weight: num  4.17 5.58 5.18 6.11 4.5 4.61 5.17 4.53 5.33 5.14 ...
 $ group : Factor w/ 3 levels "ctrl","trt1",..: 1 1 1 1 1 1 1 1 1 1 ...
head(plantas) # Conferir parte inicial do objeto
  weight group
1   4.17  ctrl
2   5.58  ctrl
3   5.18  ctrl
4   6.11  ctrl
5   4.50  ctrl
6   4.61  ctrl
tail(plantas) # Conferir parte final do objeto
   weight group
25   5.37  trt2
26   5.29  trt2
27   4.92  trt2
28   6.15  trt2
29   5.80  trt2
30   5.26  trt2
```

6.1 Escolher arquivo e modo colar

Além de carregar os arquivos diretamente da pasta de trabalho também é possível escolher um arquivo de maneira interativa usando a função *file.choose*. Outra opção é usar as opções copiar e colar do sistema operacionar usando as opções *"clipboard"* ou *pipe("pbpaste")*. Essas opções parecem ser fáceis em um primeiro momento, mas com o tempo se tornam mais complicadas, pois não é registrado qual arquivo exatamente foi usado em uma determinada rotina de análise. É preferiver carregar os arquivos diretamente usando o nome do arquivo dentro da pasta de trabalho.

```
read.table(file.choose()) # Abre uma janela para procurar o arquivo
read.table("clipboard") # Modo coloar no sistema Windows
read.table(pipe("pbpaste")) # Modo coloar no sistema Mac
```

7 Operações básicas

O R pode realizar muitas operações aritméticas, lógicas ou estatísticas de maneira muito simplificada. Essas operações podem ser aplicadas em objetos, vetores ou data.frames inteiros.

7.1 Dados de exemplo

```
ex1 <- c(4, 8.987, 48.04, 3.22, 43, 2.34) # Números positivos
ex1
[1]  4.000  8.987 48.040  3.220 43.000  2.340
ex2 <- c(-12, -8.12, 2100, 7, NA, -57.3) # Números, negativos e faltantes
ex2
[1]  -12.00   -8.12 2100.00    7.00      NA  -57.30
ex3 <- c(8, 9, 1.65, 12.8, 1, 199) # Mais números
ex3
```

```
[1]   8.00   9.00   1.65  12.80   1.00 199.00
A <- matrix(c(1,2,3, 11,12,13), nrow = 2, ncol = 3) # Matriz, 2x3
A
     [,1] [,2] [,3]
[1,]    1    3   12
[2,]    2   11   13
B <- matrix(c(12,29,10, 18,28,54, 9, 7, 35), nrow = 3, ncol = 3) # Matriz, 3x3
B
     [,1] [,2] [,3]
[1,]   12   18    9
[2,]   29   28    7
[3,]   10   54   35
```

7.2 Operações aritméticas

```
ex1+100 # Somar uma contante ao vetor
[1] 104.000 108.987 148.040 103.220 143.000 102.340
ex1^2 # Elevar ao quadrado o vetor
[1]   16.00000   80.76617 2307.84160   10.36840 1849.00000    5.47560
max(ex1) # Máximo
[1] 48.04
min(ex1) # Mínimo
[1] 2.34
sum(ex1) # Somar todos os valores do vetor
[1] 109.587
sum(ex2) # Quando existe algum NA o resultado é NA, mas...
[1] NA
sum(ex2, na.rm = TRUE) # ... é possível remover os NA da operação
[1] 2029.58
factorial(6) # Fatorial
[1] 720
log(ex1) # Logaritmo
[1] 1.3862944 2.1957791 3.8720340 1.1693814 3.7612001 0.8501509
sqrt(ex1) # Raiz quadrada
[1] 2.000000 2.997833 6.931089 1.794436 6.557439 1.529706
sqrt(ex2) # Raiz quadrada, retorna um NaN, pois não há raiz de números negativos
Warning in sqrt(ex2): NaNs produced
[1]       NaN       NaN 45.825757  2.645751        NA       NaN
abs(ex1) # Valores absoluto
[1]  4.000  8.987 48.040  3.220 43.000  2.340
A%*%B # Multiplicação de matrizes, A vezes B
     [,1] [,2] [,3]
[1,]  219  750  450
[2,]  473 1046  550
rowSums(A) # Soma das linhas
[1] 16 26
colSums(A) # Soma das colunas
[1]  3 14 25
round(ex1, digits = 1) # Arredondar número para 1 dígito decimal
[1]  4.0  9.0 48.0  3.2 43.0  2.3
sort(ex1) # Organizar em ordem crescente
[1]  2.340  3.220  4.000  8.987 43.000 48.040
```

```
sort(ex1, decreasing = TRUE) # Organizar em ordem decrescente
[1] 48.040 43.000  8.987  4.000  3.220  2.340
order(ex1) # Obter posição (não o valor!) de cada valor na ordem crescente...
[1] 6 4 1 2 5 3
ex1[order(ex1)] # ... é possível reorganizar os números conforme essa ordem obtida
[1]  2.340  3.220  4.000  8.987 43.000 48.040
```

7.3 Operações lógicas

```
valor <- 0.83 # Valor de exemplo
valor
[1] 0.83
valor < 0.9 # Menor (> para maior)
[1] TRUE
valor <= 0.83 # Menor ou igual (>= para maior ou igual)
[1] TRUE
valor == 9 # valor exatamente igual
[1] FALSE
valor != 0.8 # valor diferente de
[1] TRUE
ex1 == 4 # Podem ser aplicados a vetores
[1]  TRUE FALSE FALSE FALSE FALSE FALSE
0<=valor & valor<=0.1 # Dois criterios aditivos
[1] FALSE
0<=valor | valor<=0.1 # Dois criterios, um ou outro
[1] TRUE
!valor == 0.83 # Inverter o argumento lógico
[1] FALSE
ifelse(ex1>4, yes = "M", no = "F") # Teste condicional para categorizar valores
[1] "F" "M" "M" "F" "M" "F"
```

7.4 Estatística descritiva

```
mean(ex1) # Média
[1] 18.2645
median(ex1) # Mediana
[1] 6.4935
var(ex1) # Variância
[1] 453.58
sd(ex1) # Desvio padrão
[1] 21.29742
cor(ex1, ex3) # Correlação entre ex1 e ex3
[1] -0.415766
cor(ex1, ex3, method = "spearman") # Correlação entre ex1 e ex3 usando Spearman
[1] -0.8857143
cov(ex1, ex3) # Covariação entre ex1 e ex3
[1] -697.0587
```

7.5 Gerar números, sequências, dados aleatórios e amostragem

Muitas funções que auxiliam na geração de sequências númericas ou de caracteres, extrai números de distribuições estatísticas e realizar procedimentos de amostragem.

```
1:10 # Sequência de 1 a 10
 [1]  1  2  3  4  5  6  7  8  9 10
-9:1 # Sequência de -9 a 1
 [1] -9 -8 -7 -6 -5 -4 -3 -2 -1  0  1
seq(from = 1, to = 10, by = 2) # Sequência regulare de 1 a 10 por 2
[1] 1 3 5 7 9
seq(from = 1, to = 3, length.out = 8) # Sequência regulare de 1 a 3, com 8 valores
[1] 1.000000 1.285714 1.571429 1.857143 2.142857 2.428571 2.714286 3.000000
seq_len(12) # Sequência de inteiros de 1 a 12
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
paste("A", 1:6, sep = ".") # Sequência regular de caracteres
[1] "A.1" "A.2" "A.3" "A.4" "A.5" "A.6"
rep(3, 6) # Repetir o valor 3 por 6 vezes
[1] 3 3 3 3 3 3
rep(1:4, 3) # Repetir a sequência 1 a 4 por 3 vezes
 [1] 1 2 3 4 1 2 3 4 1 2 3 4
rep(1:4, each = 3) # Repetir os valores 1 a 4 por 3 vezes cada
 [1] 1 1 1 2 2 2 3 3 3 4 4 4
runif(5, min = 0, max = 10) # Gerar distribuição uniforme, entre 0 e 10...
[1] 0.7580746 2.2462758 5.0755553 4.6462726 2.5227444
runif(5, min = 0, max = 10) # ... Note que é diferente a cara execução
[1] 4.665603 9.498394 7.177170 7.633072 4.992322
rnorm(5, mean = 0, sd = 1) # Gerar distribução normal com média 0 e variância 1
[1]  0.7060718 -0.7819301 -0.6400494  0.5360680  0.1856078
rpois(5, lambda = 1) # Gerar distribuição de Poisson, com lambda 1
[1] 0 1 1 1 0
sample(1:10, size = 4) # Amostrar 4 valores da sequência de 1 a 10 sem reposição...
[1] 4 2 3 8
sample(1:10, size = 4, replace = TRUE) # ... Amostrar com reposição
[1] 10  7  4  3
moeda <- c("cara","coroa") # Vetor com resultados de uma moeda
moeda
[1] "cara"  "coroa"
sample(moeda, size = 1) # Amostrar um resultado
[1] "coroa"
sample(moeda, size = 3, replace = TRUE) # Amostrar 3 vezes, replace deve ser verdadeiro
[1] "cara"  "cara"  "coroa"
```

7.6 Aplicar funções em vetores ou tabela de dados

As funções básicas aplicadas a vetores também podem ser aplicadas facilmente a data.frames, matrizes ou listas. Essas funções requerem bastante atenção na definição dos parâmetros por serem bastante versáteis e facilitam aplicação de cálculos mais complexos. Algumas deles são listadas a seguir.

```
sapply(eidat, sum) # Aplicar função sum em cada vetor do data.frame/listas
   x0    x1    t0    t1
 1948 22100  6970 17078
apply(eidat, MARGIN = 2, sum) # Aplicar função sum matrizes/data.frames...
   x0    x1    t0    t1
 1948 22100  6970 17078
# ... Quando MARGIN é 2 por colunas, quando 1 é por linhas
apply(eidat, MARGIN = 1, sd) # Aplicar função sd matrizes/data.frames
       1        2        3        4        5        6        7        8        9
```

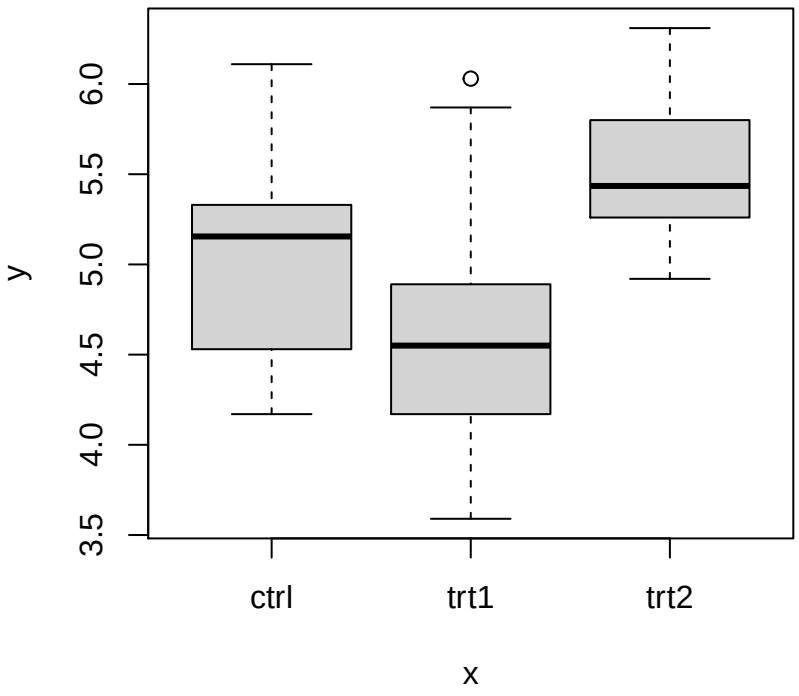
```
1648.0515 1016.6392  941.1670  679.5059  274.9497  978.7238 1867.1865  357.5439 1115.9261
       10
1455.2206
replicate(n = 3, expr = sample(moeda, size = 1)) # Replicar função/expressão por 3 vezes
[1] "coroa" "cara"  "coroa"
tapply(plantas$weight, INDEX = plantas$group, FUN = mean) # Aplicar mean por plantas$group
 ctrl  trt1  trt2
5.032 4.661 5.526
table(plantas$group) # Tabela de contingência das contagens em cada categoria

ctrl trt1 trt2
  10   10   10
```

8 Gráficos

A função *plot* é uma função genérica que produz gráficos de maneira automático dependendo do tipo de informação e classe do objeto passado para ela. Quando apropriado, os eixos, nomes dos eixos e títulos são gerados automaticamente. Existem diversas funções auxiliares para adicionar elementos aos gráficos. Um texto mais completo sobre gráficos pode ser encontrado em https://vanderleidebastiani.github.io/tutoriais.

```
plot(plantas$group, plantas$weight)
```



9 Exportar resultados

Assim como para importar dados, para exportar há muitas funções para exportar dados. Para exportar tabelas completas é possível usar as funções *write.table* ou *write.csv*, basta especificar o objeto com a tabela completa e o nome do arquivo de saída. O arquivo será salvo no diretório de trabalho do R. Para exportar os resultados de análises para visualização é melhor usar o *Rmarkdown* (um texto básico pode ser encontrado em https://vanderleidebastiani.github.io/tutoriais).

```r
write.table(dados, "dados.txt", sep = " ") # Exportar em .txt
write.csv(dados, "dados.csv") # Exportar em .csv
```

10 Guia de ajuda rápida

```
# - Adicionar comentário
? - Obter ajuda de função
?? - Relizar buscar
<- ou = - Atribuir objeto (direita para esquerda)
-> - Atribuir objeto (esquerda para direita)
+ - Somar
- - Subtrair
* - Multiplicar
/ - Dividir
^ - Potêncializar (direita para esquerda)
%*% - Multiplicar matrizes
< - Comparar, menor
> - Comparar, maior
<= - Comparar, menor ou igual
>= - Comparar, maior ou igual
== - Comparar, exatamente igual
!= - Comparar, diferente
! - Lógico, NÃO. Inverter resultado de teste lógico
& - Lógico, critério aditivo E. Operação elementar
| - Lógico, critério aditivo OU. Operação elementar
&& - Lógico, E
|| - Lógico, OU
~ - Fórmula estatística
FALSE ou F - Argumento lógico falso
TRUE ou T - Argumento lógico verdadeiro
NA - Indeterminado (Not Avaiable)
NaN - Indeterminado (Not a Number)
Inf - Infinito
NULL - Objeto nulo
c() - Concatenar valores em vetor
factor() - Criar fator
ordered() - Criar fator ordenado
data.frame() - Criar tabela de dados (data.frames)
matrix() - Criar matriz
list() - Criar lista
rbind() - Combinar vetores por linhas
cbind() - Combinar vetores por colunas
paste() - Concatenar caracteres em sequências regulares
class() - Conferir/Atribuir classe do objeto
str() - Conferir estrutura do objeto
: - Gerar sequência numérica contínua
$ - Indexar vetores/listas pelo nome das variáveis/listas
@ - Indexar na classe S4
[] - Indexar vetores/listas
[ , , drop = FALSE] - Indexar data.frames/matrizes. Primeiro valor linha, segundo coluna
[[]] - Indexar listas
names() - Conferir/Atribuir nomes a vetores/listas
colnames() - Conferir/Atribuir nomes as linhas de data.frames/matrizes
```

```
rownames() - Conferir/Atribuir nomes as colunas de data.frames/matrizes
length() - Conferir comprimento de vetores/listas
dim() ou nrow() e ncol() - Conferir dimensões de data.frames/matrizes
head() e tail() - Conferir início e fim de data.frames/matrizes
t() - Transpor data.frames/matrizes
seq() - Obter sequência regular
rep() - Repetir valores
ifelse() - Aplicar teste condicional
read.csv() ou read.table() - Importar tabelas
write.csv() ou write.table() - Exportar tabelas
ls() - Listar objetos da área de trabalho
rm() - Remover objetos da área de trabalho
save.image() - Salvar área de trabalho
load() - Carregar área de trabalho
max() - Obter máximo
min() - Obter mínimo
sum() - Obter soma
sqrt() - Obter raiz quadrada
log() - Obter logaritmo
exp() - Obter função exponential
abs() - Obter valores absoluto
round() - Arredondar valores
factorial() - Obter fatorial
mean() - Obter média
median() - Obter mediana
var() - Obter variância
sd() - Obter desvio padrão
cor() - Obter correlação
cov() - Obter covariação
runif() - Gerar distribuição uniforme
rnorm() - Gerar distribução normal
rpois() - Gerar distribuição de Poisson
sample() - Amostrar valores
sort() - Ordenar valores
order() - Obter posição da order dos valores
which() - Procurar posição (índice) conforme teste lógico
sapply() - Aplicar função em data.frames/listas
tapply() - Aplicar função em vetores
apply() - Aplicar função em data.frames/matrizes
replicate() - Replicar expressões
table() - Produzir tabelas de contingência
```

11 Conclusão

O objetivo deste texto foi introduzir alguns conceitos básicos da linguagem R. Os principais aspectos da sintaxe, operadores, tipos de dados e estruturas dos objetos foram mostradas e brevemente exemplificados. Espero que este texto tenha sido útil e, por favor, avise-me se tiver dúvidas ou sugestões sobre este texto.

12 Mais informações

Outros textos e tutoriais sobre R podem ser encontrados em https://vanderleidebastiani.github.io/tutoriais.

13 Referências

- Crawley, Michael J. 2007. **The R book**. John Wiley & Sons, Chichester.
- R Core Team; 2018. **R Language Definition**. https://cran.r-project.org/doc/manuals/R-lang.html